Visualizing the Graphical Execution of Programs

for JavaScript Abstract Interpretation

Jane Hoffswell

Harvey Mudd College class of 2014

How can we better understand programs?

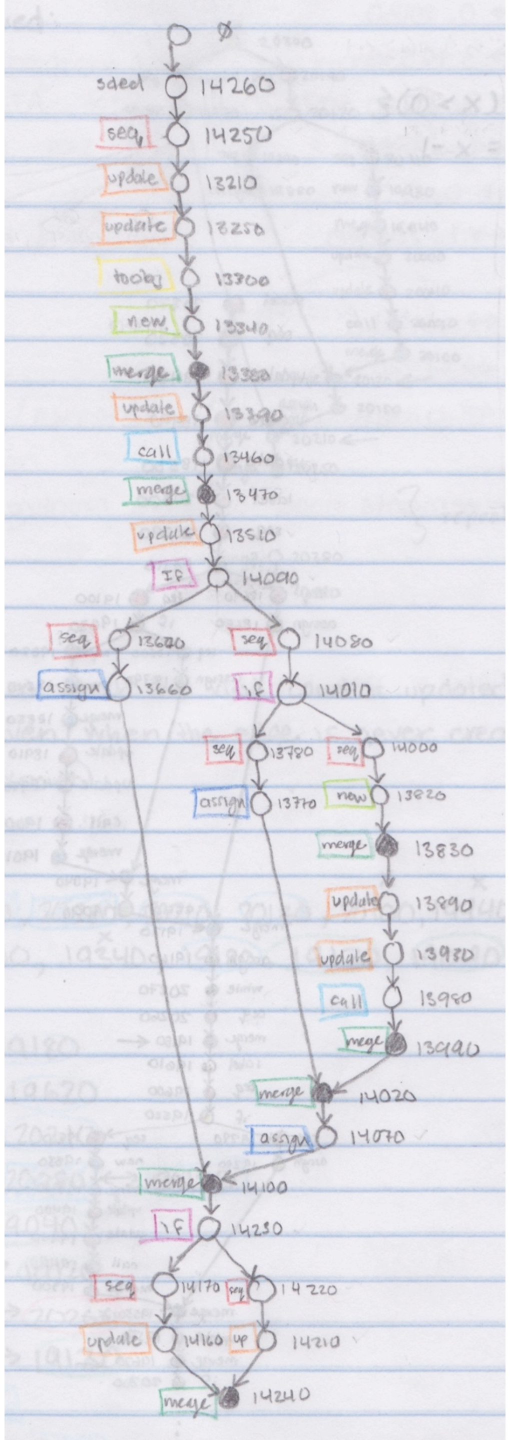
```
x = Math.random();
if (x < 0.5)
    y = true;
else
    y = false;
```

```
x = Math.random();
if (x < 0.5)
    y = true;
else
    y = false;
```

```
SDecl                                    ID: 14180
   Seq                                   ID: 14170
     Update                              ID: 13130
       Var: `window`0 (0)                ID: 13100
       String: dummyAddress              ID: 13110
       Undef                             ID: 13120
     Update                              ID: 13170
       Var: `window`0 (0)                ID: 13140
       String: Arguments                 ID: 13150
       Undef                             ID: 13160
     ToObj                               ID: 13220
       Scratch: 0                        ID: 13180
       Binop: acc                        ID: 13210
         Var: `window`0 (0)              ID: 13190
         String: Math                    ID: 13200
     New                                 ID: 13260
       Scratch: 1                        ID: 13230
       Var: `argumentsVar`11 (11)        ID: 13240
       Var: `dummyAddressVar`13 (13)     ID: 13250
```

```
x = Math.random();
if (x < 0.5)
     y = true;
else
     y = false;
```

```
SDecl                                        ID: 14180
   Seq                                       ID: 14170
     Update                                  ID: 13130
       Var: `window`0 (0)                    ID: 13100
       String: dummyAddress                  ID: 13110
       Undef                                 ID: 13120
     Update                                  ID: 13170
       Var: `window`0 (0)                    ID: 13140
       String: Arguments                     ID: 13150
       Undef                                 ID: 13160
     ToObj                                   ID: 13220
       Scratch: 0                            ID: 13180
       Binop: acc                            ID: 13210
         Var: `window`0 (0)                  ID: 13190
         String: Math                        ID: 13200
     New                                     ID: 13260
       Scratch: 1                            ID: 13230
       Var: `argumentsVar`11 (11)            ID: 13240
       Var: `dummyAddressVar`13 (13)         ID: 13250
```



```
function fact(n) {
    if (n <= 0) return 1;
    else return n*fact(n-1);
}

var btop = (fact(3) === 6);

print(btop);

var CObject = {
    results: {
        FAIL: "failure"
    },
    valid: true
};

var fail = CObject.results.FAIL;

print(fail);

var foo = {
    Qi: function(aid) {
        if (aid) {
            return this;
        }
        else throw CObject.results.FAIL;
    },

    olchange: function(p) {
        var isValid = btop? CObject.valid: p;
        CObject.reverse = !isValid;
    }
}

try {
    foo.Qi(btop).olchange(false);
    print(CObject.reverse);
    foo.Qi(btop).bar = 42;
    print(foo.bar);
} catch (x) {
    print("Caught");
}
```

?

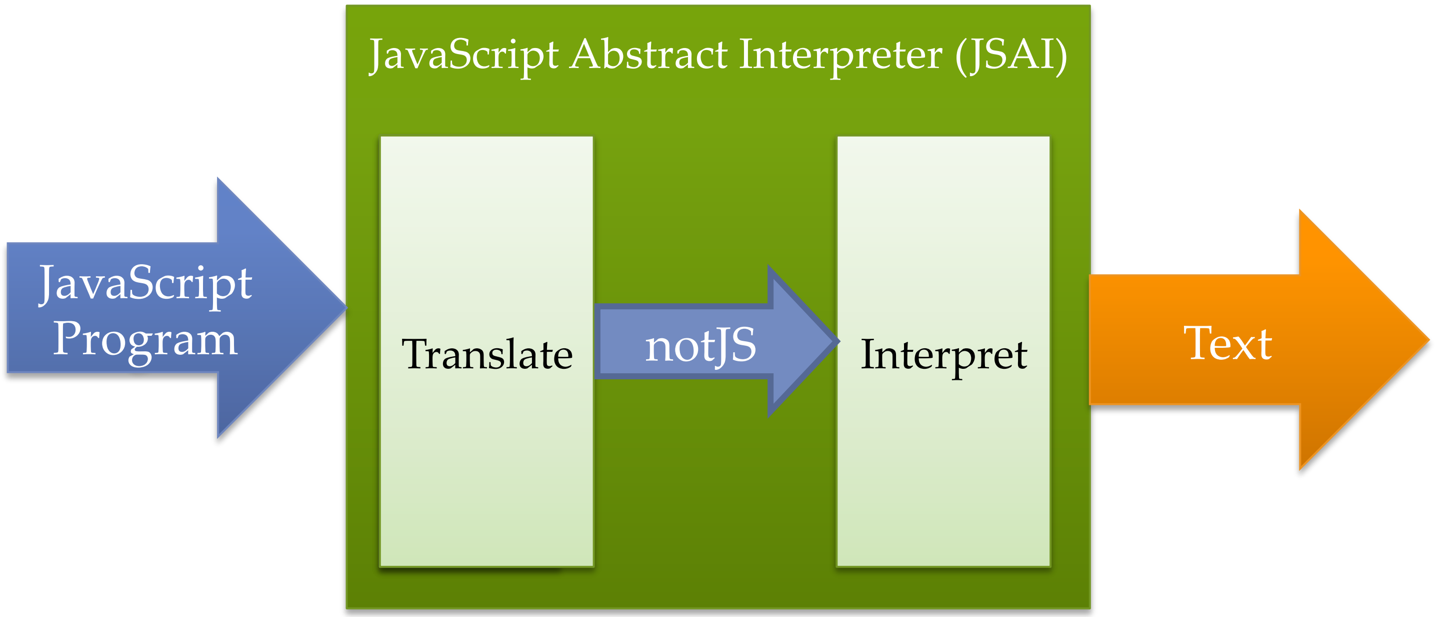
```
function fact(n) {
    if (n <= 0) return 1;
    else return n*fact(n-1);
}

var btop = (fact(3) === 6);

print(btop);

var CObject = {
    results: {
        FAIL: "failure"
    },
    valid: true
};

var fail = CObject.results.FAIL;

print(fail);

var foo = {
    Qi: function(aid) {
        if (aid) {
            return this;
        }
        else throw CObject.results.FAIL;
    },

    olchange: function(p) {
        var isValid = btop? CObject.valid: p;
        CObject.reverse = !isValid;
    }
}

try {
    foo.Qi(btop).olchange(false);
    print(CObject.reverse);
    foo.Qi(btop).bar = 42;
    print(foo.bar);
} catch (x) {
    print("Caught");
}
```

GNU Debugger

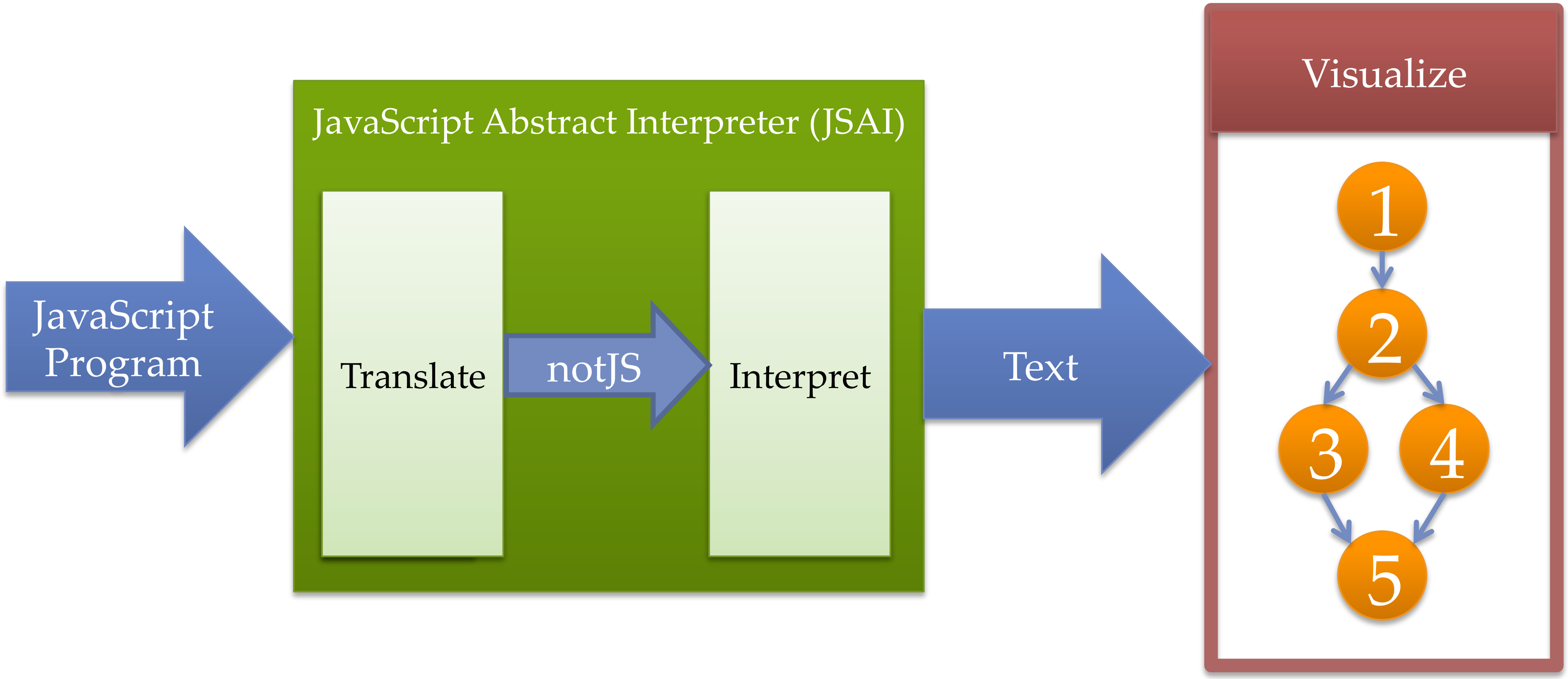
Valgrind

Print Statements

Inspect Output



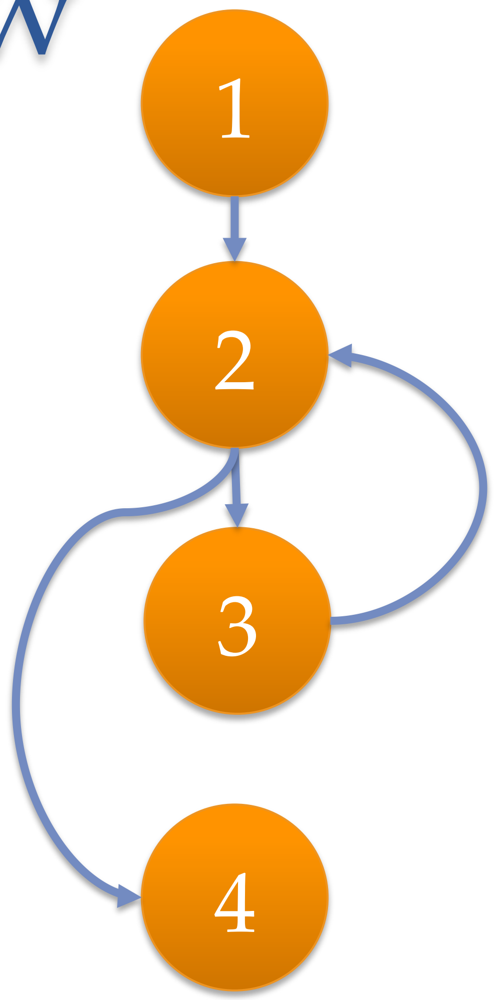
```
…
=> {FSCI(58250)}
=> {FSCI(58170),F
=> {FSCI(58230)}
=> {FSCI(58260)}
=> {FSCI(58160)}
=> {}
=> {FSCI(56360)}
=> {FSCI(56400)}
=> {FSCI(58890)}
=> {FSCI(58930)}
=> {FSCI(58930)}
=> {FSCI(58940)}
60180: DNum:NT
Address(-61), Ad
Address(-45), Ad
Address(-58), Ad
Address(-14), Ad
Address(-72))|DU
[success] Total tin
```


JavaScript Program
JavaScript Abstract Interpreter (JSAI)
Translate
notJS
Interpret
Text
Visualize
1
2
3
4
5

Background

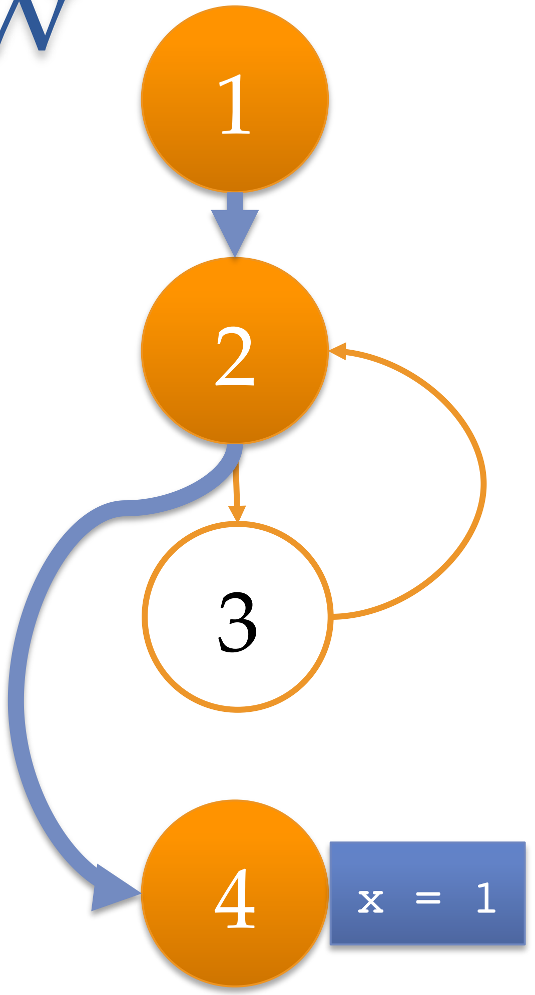
Control Flow

```
1   ...
2   while(x.isEven)
3     x = x + 1
4   print(x)
```



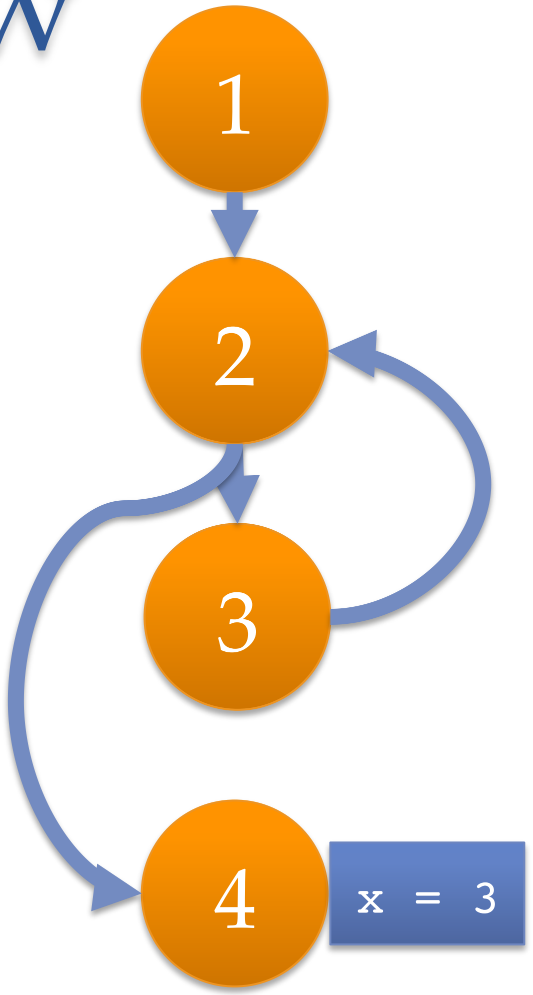
Control Flow

```
1   x = 1
2   while(x.isEven)
3     x = x + 1
4   print(x)
```



Control Flow

```
1   x = 2
2   while(x.isEven)
3     x = x + 1
4   print(x)
```

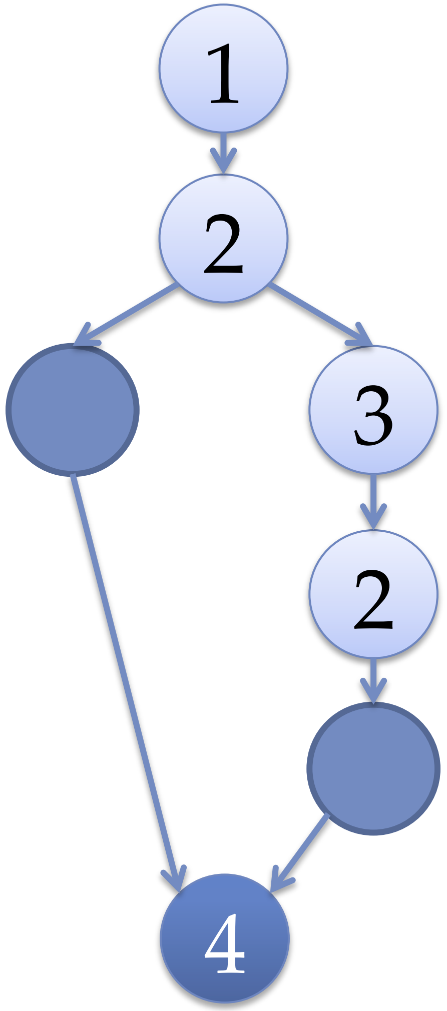


Static Analysis

```
x = Math.random()
while(x.isEven)
  x = x + 1
print(x)
```

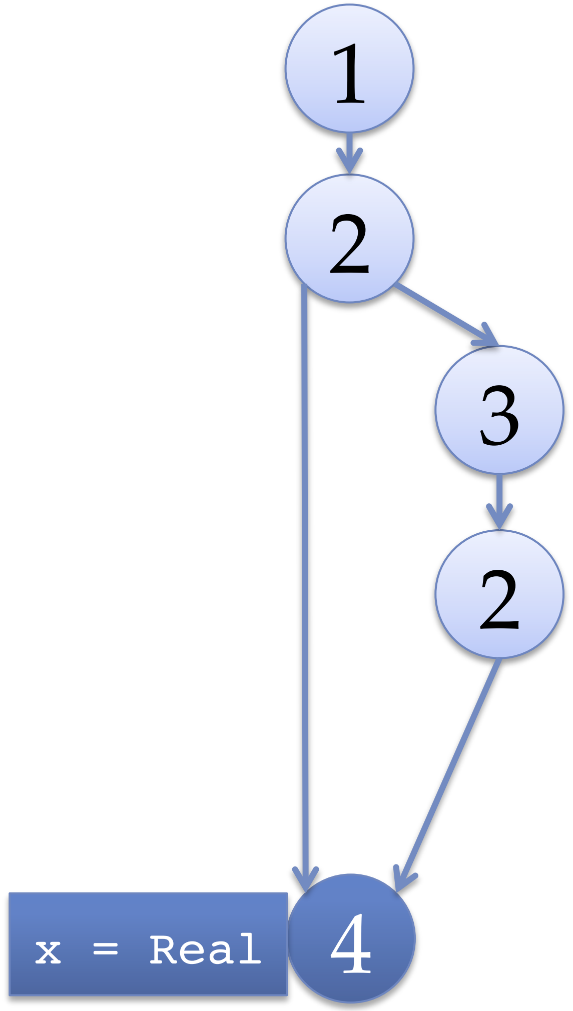
Static Analysis

```
1   x = Math.random()
2   while(x.isEven)
3     x = x + 1
4   print(x)
```



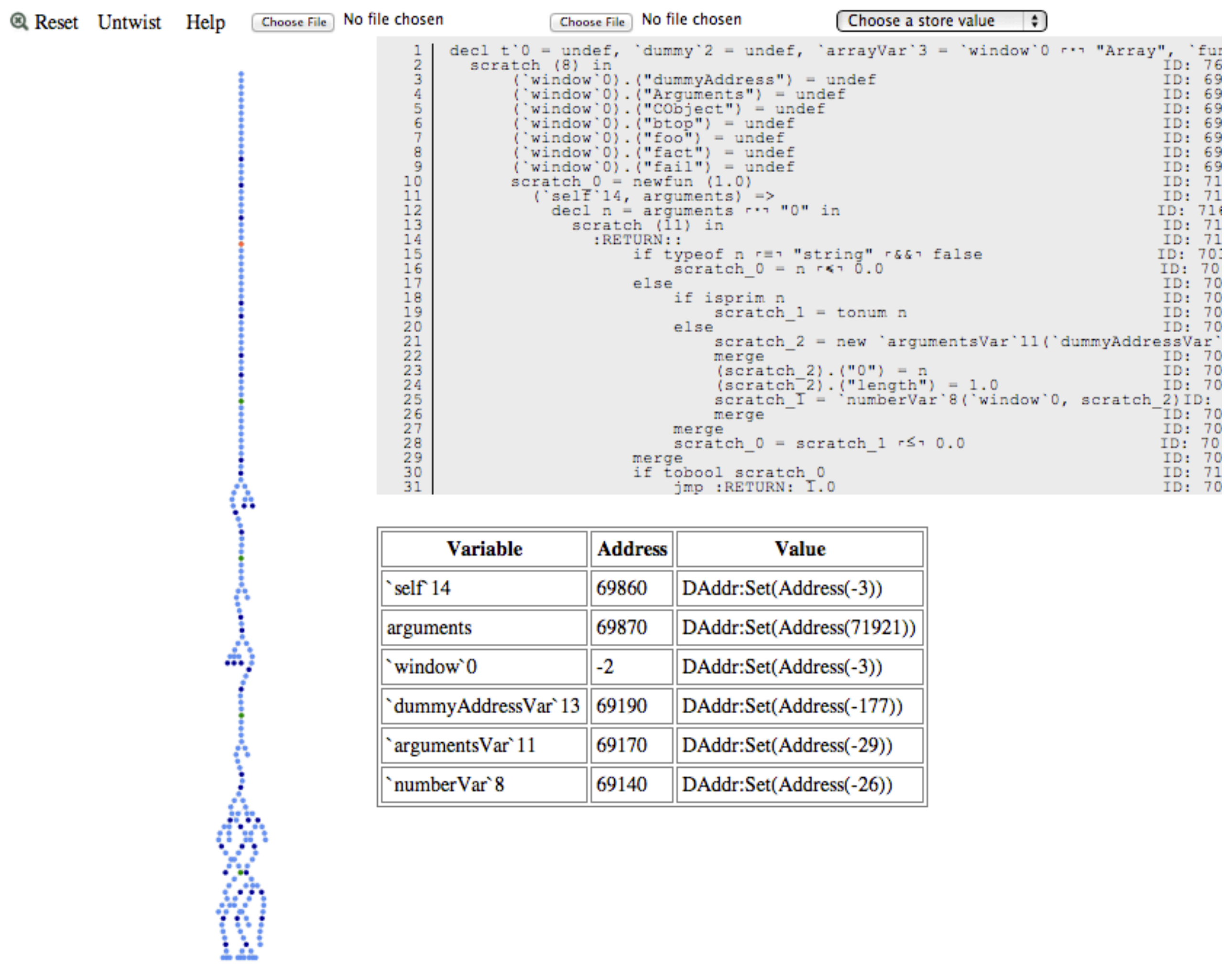
Static Analysis

```
1   x = Math.random()
2   while(x.isEven)
3     x = x + 1
4   print(x)
```



Which analyses are the most precise?

JSAI Visualizer



```
decl t`0 = undef, `dummy`2 = undef, `arrayVar`3 = `window`0 ⌜·⌝ "Array", `fu
  scratch (8) in                                                      ID: 76
      (`window`0).("dummyAddress") = undef                            ID: 69
      (`window`0).("Arguments") = undef                               ID: 69
      (`window`0).("CObject") = undef                                 ID: 69
      (`window`0).("btop") = undef                                    ID: 69
      (`window`0).("foo") = undef                                     ID: 69
      (`window`0).("fact") = undef                                    ID: 69
      (`window`0).("fail") = undef                                    ID: 69
      scratch_0 = newfun (1.0)                                        ID: 71
        (`self`14, arguments) =>                                      ID: 71
          decl n = arguments ⌜·⌝ "0" in                              ID: 71
            scratch (11) in                                           ID: 71
              :RETURN::                                               ID: 71
                  if typeof n ⌜≡⌝ "string" ⌜&&⌝ false                ID: 70
                      scratch_0 = n ⌜≮⌝ 0.0                          ID: 70
                  else                                                ID: 70
                      if isprim n                                     ID: 70
                          scratch_1 = tonum n                         ID: 70
                      else                                            ID: 70
                          scratch_2 = new `argumentsVar`11(`dummyAddressVar`
                          merge                                       ID: 70
                          (scratch_2).("0") = n                       ID: 70
                          (scratch_2).("length") = 1.0                ID: 70
                          scratch_1 = `numberVar`8(`window`0, scratch_2)ID:
                          merge                                       ID: 70
                      merge                                           ID: 70
                      scratch_0 = scratch_1 ⌜≤⌝ 0.0                  ID: 70
                  merge                                               ID: 70
                  if tobool scratch_0                                 ID: 71
                      jmp :RETURN: 1.0                                ID: 70
```

Variable	Address	Value
`self`14	69860	DAddr:Set(Address(-3))
arguments	69870	DAddr:Set(Address(71921))
`window`0	-2	DAddr:Set(Address(-3))
`dummyAddressVar`13	69190	DAddr:Set(Address(-177))
`argumentsVar`11	69170	DAddr:Set(Address(-29))
`numberVar`8	69140	DAddr:Set(Address(-26))

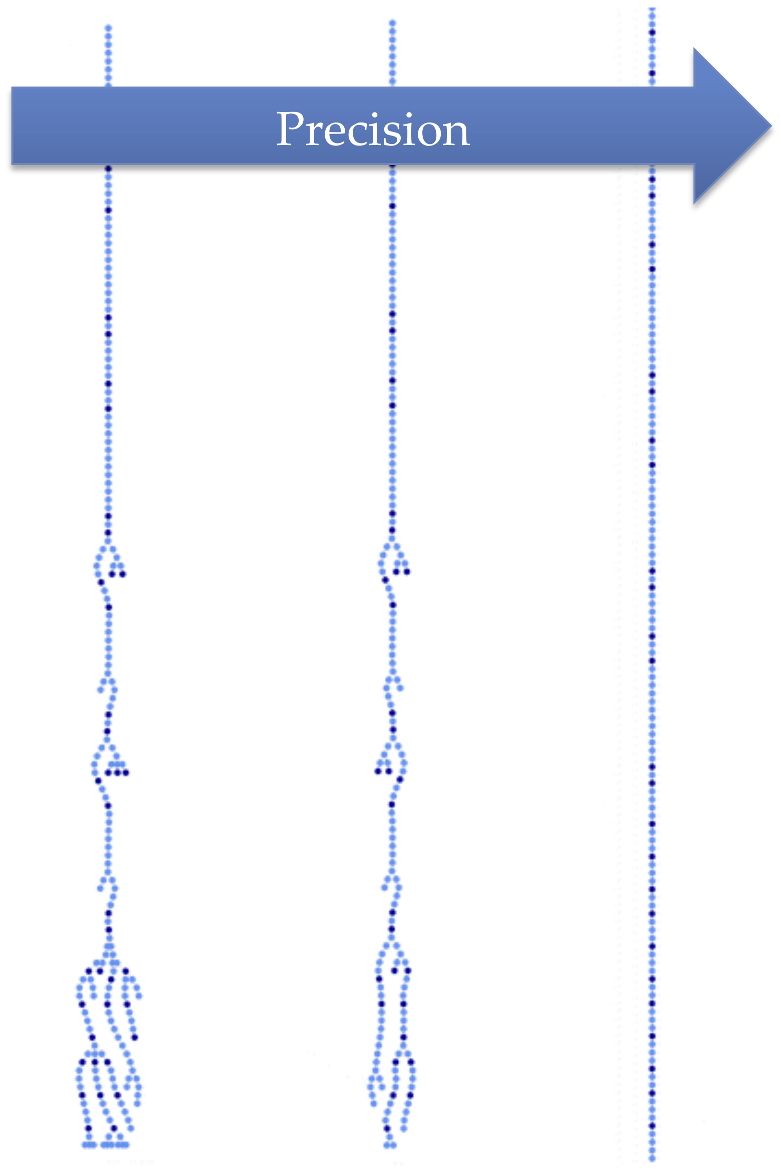
Precision & Performance

Trace Option	Runtime (sec)	File Size
Imprecise (1 CFA)	2	194 KB
Somewhat Precise (Context Insensitive)	2	151 KB
Precise (5 CFA)	2	146 KB

Table 1: v101.js

Trace Option	Runtime (sec)	File Size
Imprecise (1 CFA)	-	-
Somewhat Precise (Context insensitive)	556	3.5 MB
Precise (5 CFA)	15	741 KB

Table 2: linq_dictionary.js

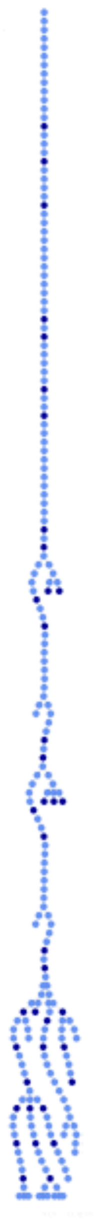


Imprecise
(1 CFA)

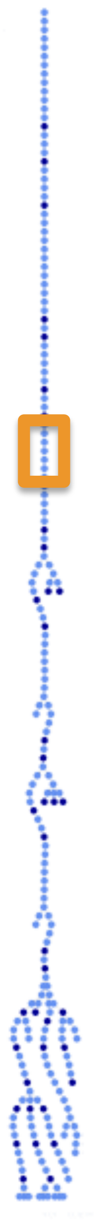
Somewhat Precise
(Context Insensitive)

Precise
(5 CFA)

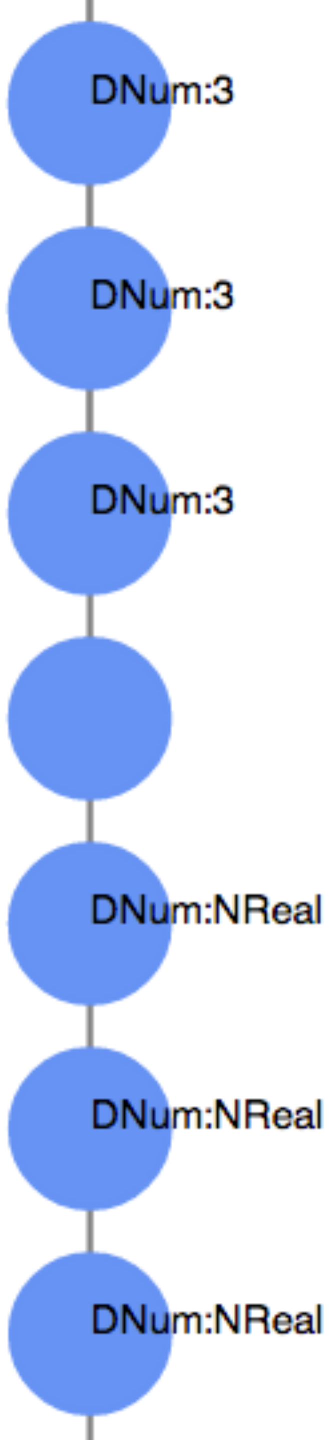
Where is precision lost?

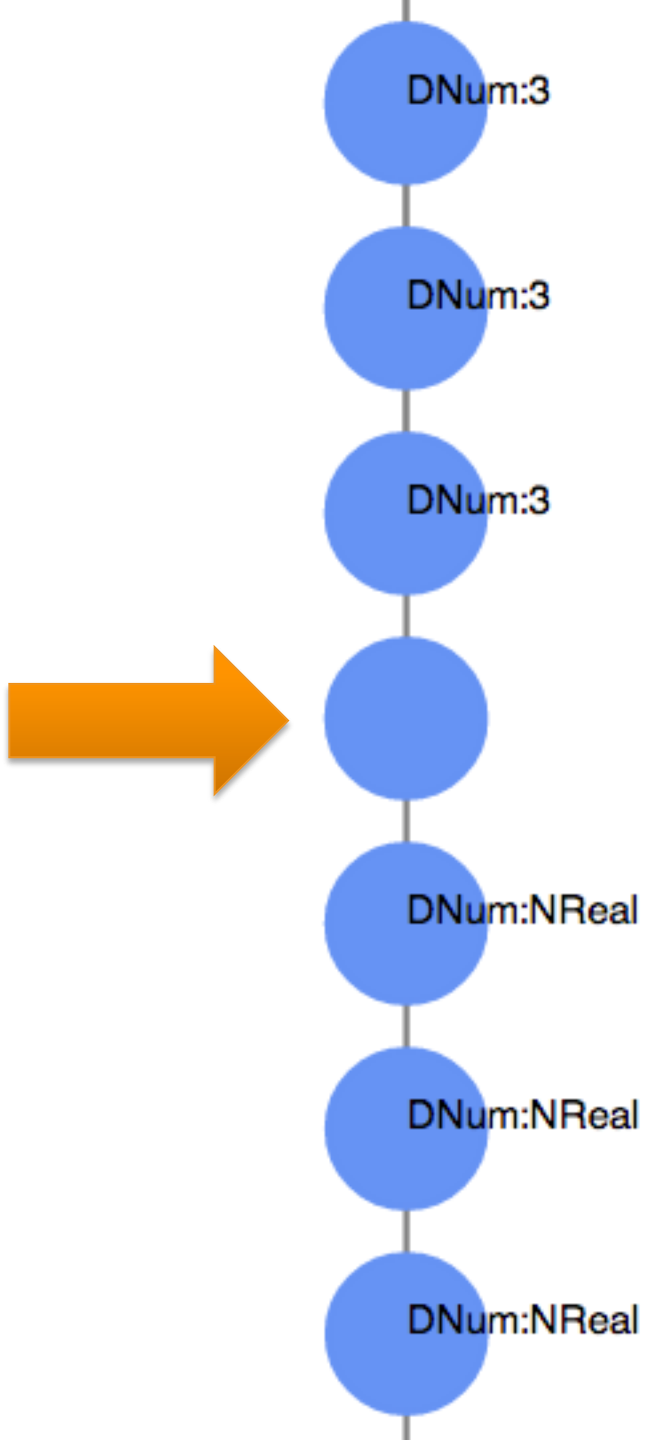


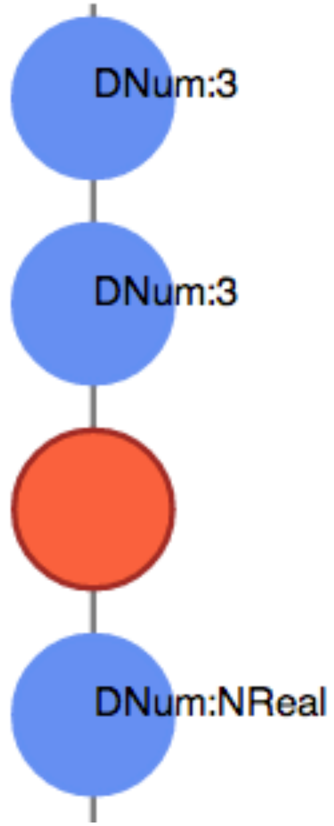
Consider the imprecise graph…

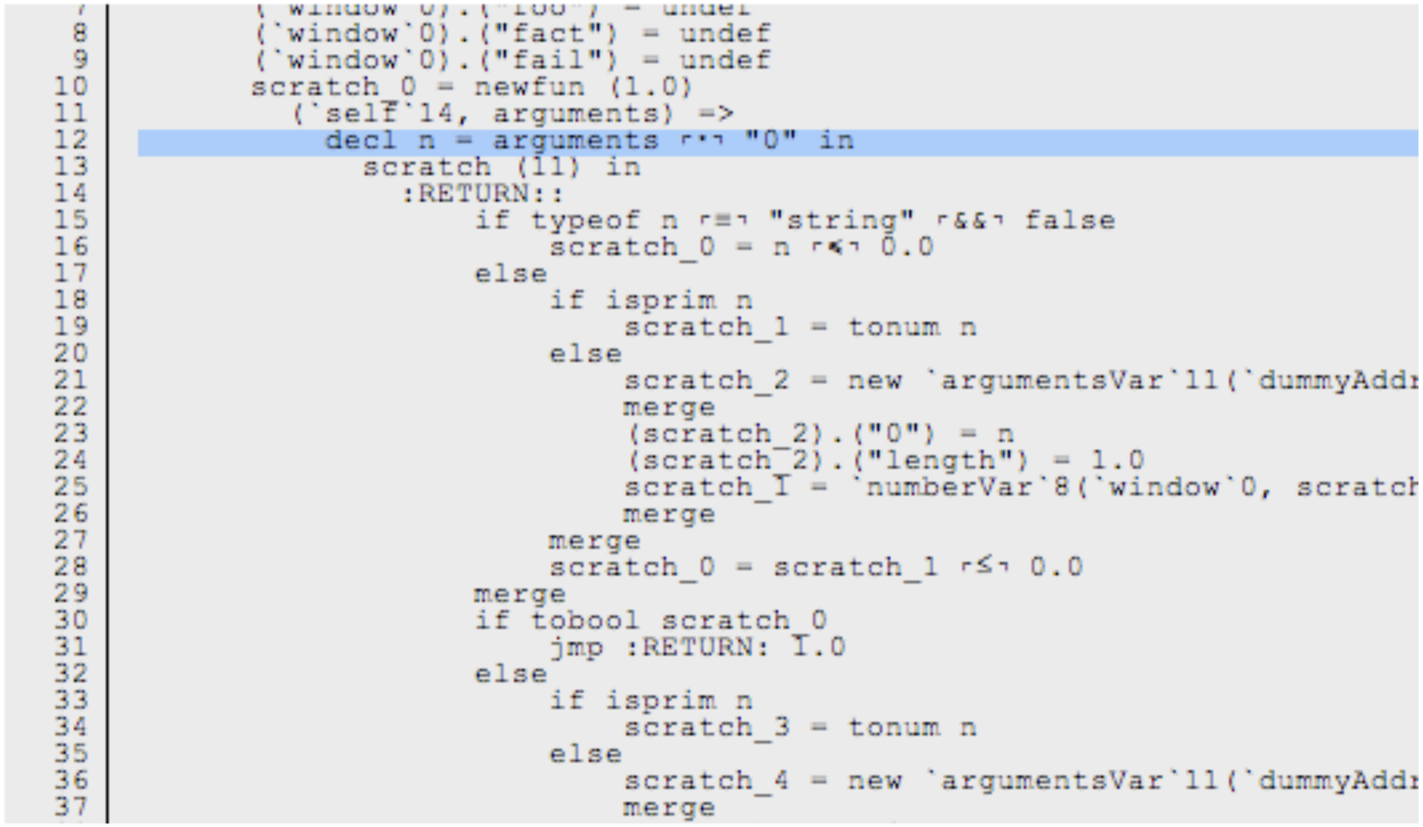


Consider the imprecise graph…

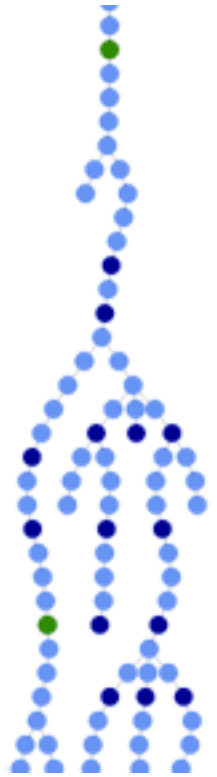

DNum:3
DNum:3
DNum:3
DNum:NReal
DNum:NReal
DNum:NReal

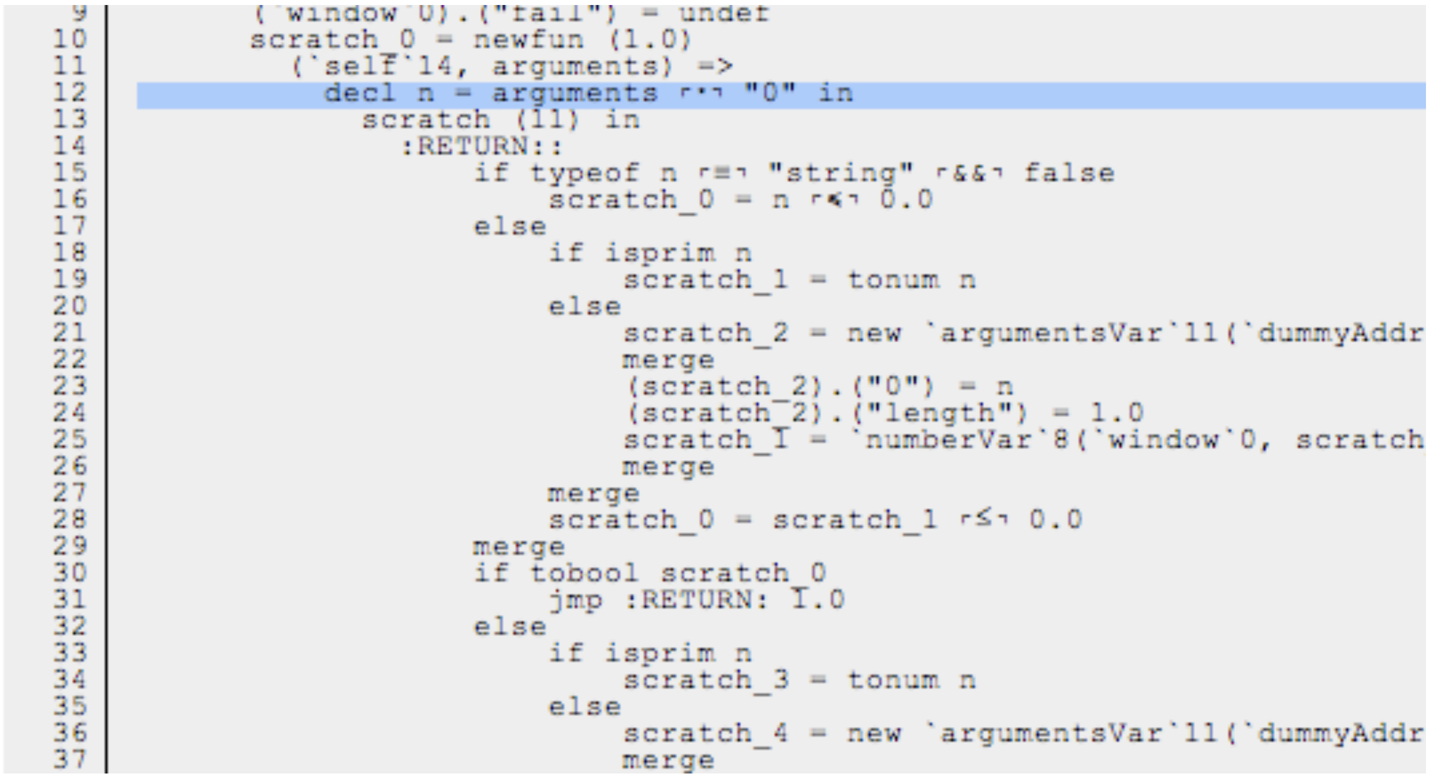

DNum:3
DNum:3
DNum:3
DNum:NReal
DNum:NReal
DNum:NReal



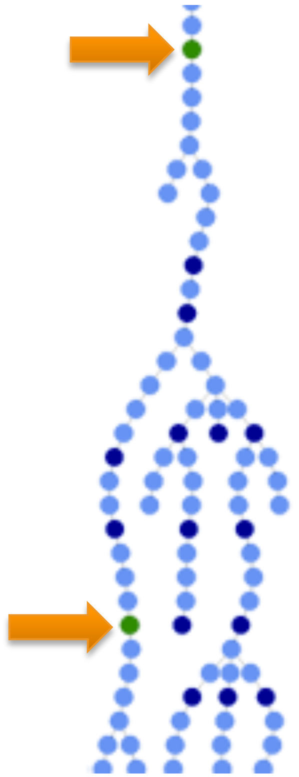


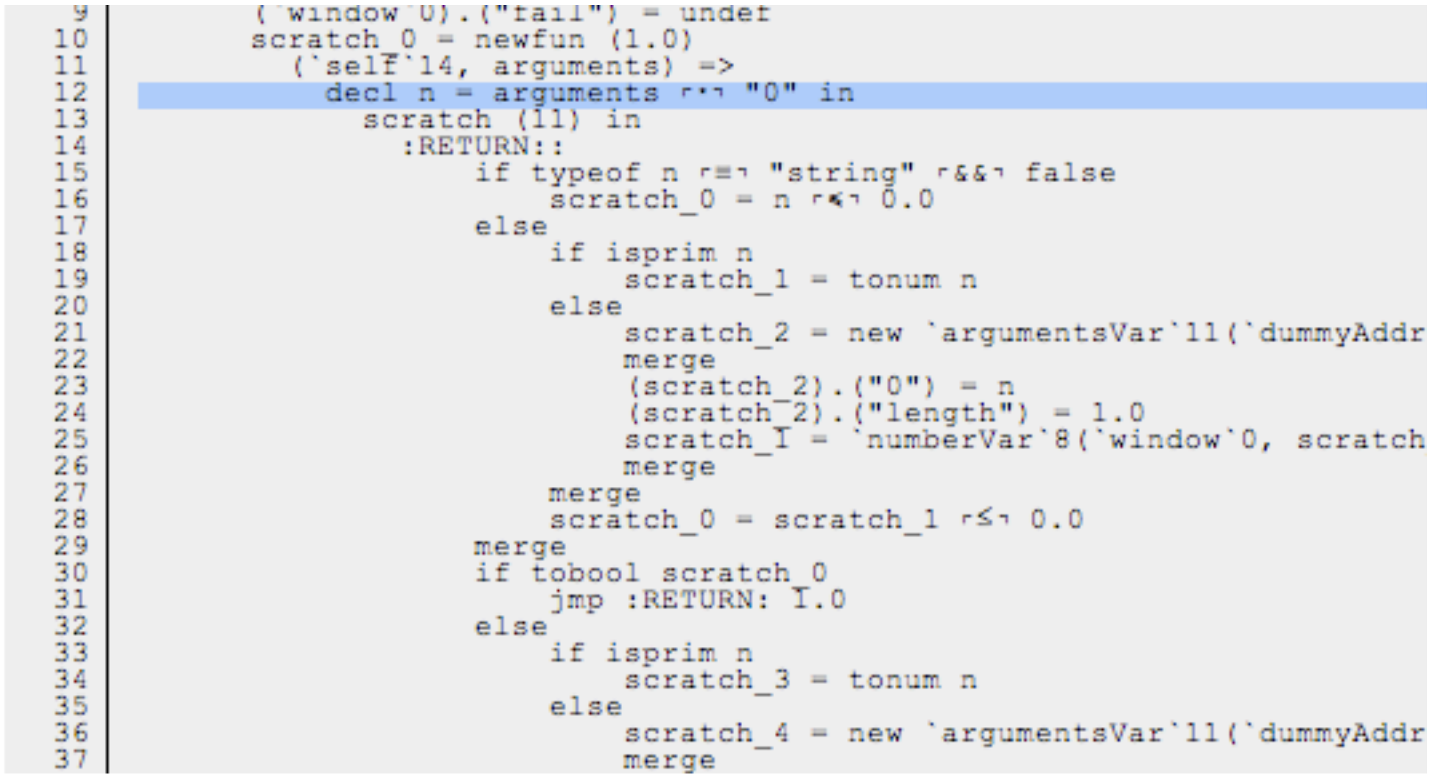
```
(`window`0).("foo") = undef
(`window`0).("fact") = undef
(`window`0).("fail") = undef
scratch_0 = newfun (1.0)
  (`self`14, arguments) =>
    decl n = arguments ⌜·⌝ "0" in
      scratch (11) in
        :RETURN::
            if typeof n ⌜=⌝ "string" ⌜&&⌝ false
                scratch_0 = n ⌜<⌝ 0.0
            else
                if isprim n
                    scratch_1 = tonum n
                else
                    scratch_2 = new `argumentsVar`11(`dummyAddr
                    merge
                    (scratch_2).("0") = n
                    (scratch_2).("length") = 1.0
                    scratch_1 = `numberVar`8(`window`0, scratch
                    merge
                merge
                scratch_0 = scratch_1 ⌜≤⌝ 0.0
            merge
            if tobool scratch_0
                jmp :RETURN: 1.0
            else
                if isprim n
                    scratch_3 = tonum n
                else
                    scratch_4 = new `argumentsVar`11(`dummyAddr
                    merge
```





```
(`window`0).("fail") = undef
scratch_0 = newfun (1.0)
  (`self`14, arguments) =>
    decl n = arguments ⌜⋅⌝ "0" in
      scratch (11) in
        :RETURN::
            if typeof n ⌜≡⌝ "string" ⌜&&⌝ false
                scratch_0 = n ⌜<⌝ 0.0
            else
                if isprim n
                    scratch_1 = tonum n
                else
                    scratch_2 = new `argumentsVar`11(`dummyAddr
                    merge
                    (scratch_2).("0") = n
                    (scratch_2).("length") = 1.0
                    scratch_1 = `numberVar`8(`window`0, scratch
                    merge
                merge
                scratch_0 = scratch_1 ⌜≤⌝ 0.0
            merge
            if tobool scratch_0
                jmp :RETURN: 1.0
            else
                if isprim n
                    scratch_3 = tonum n
                else
                    scratch_4 = new `argumentsVar`11(`dummyAddr
                    merge
```

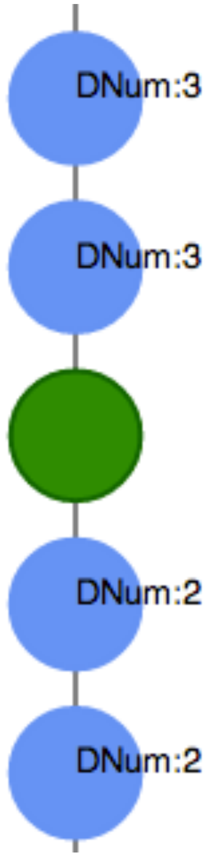


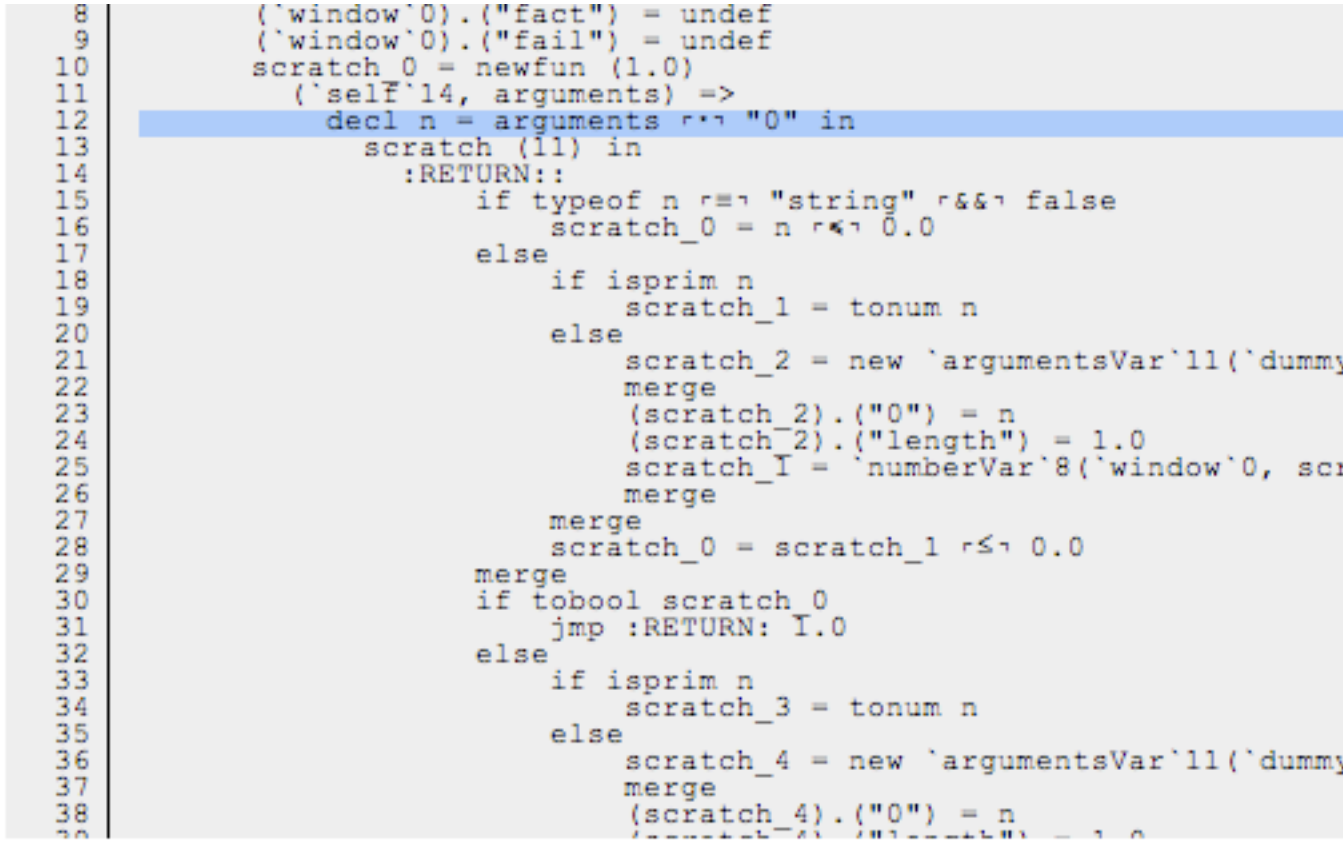


```
(`window`0).("fail") = undef
scratch_0 = newfun (1.0)
  (`self`14, arguments) =>
    decl n = arguments ⌜·⌝ "0" in
      scratch (11) in
        :RETURN::
            if typeof n ⌜≡⌝ "string" ⌜&&⌝ false
                scratch_0 = n ⌜≤⌝ 0.0
            else
                if isprim n
                    scratch_1 = tonum n
                else
                    scratch_2 = new `argumentsVar`11(`dummyAddr
                    merge
                    (scratch_2).("0") = n
                    (scratch_2).("length") = 1.0
                    scratch_1 = `numberVar`8(`window`0, scratch
                    merge
                merge
                scratch_0 = scratch_1 ⌜≤⌝ 0.0
            merge
            if tobool scratch_0
                jmp :RETURN: 1.0
            else
                if isprim n
                    scratch_3 = tonum n
                else
                    scratch_4 = new `argumentsVar`11(`dummyAddr
                    merge
```

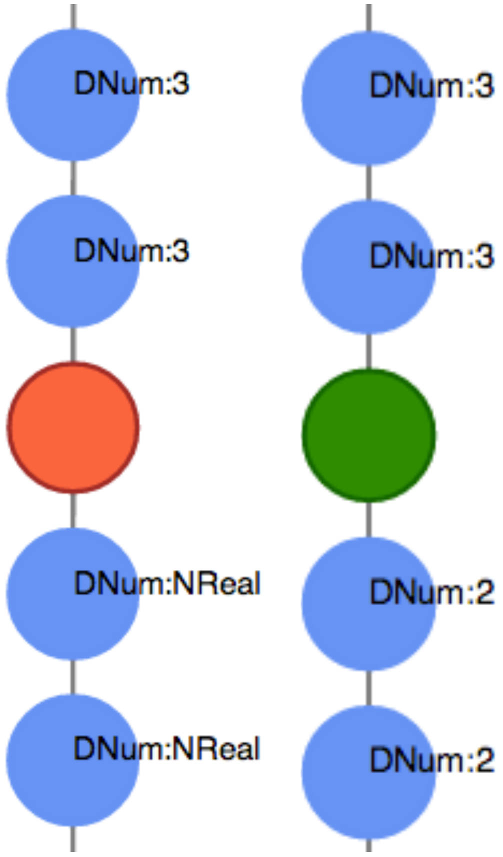
Consider the precise graph…

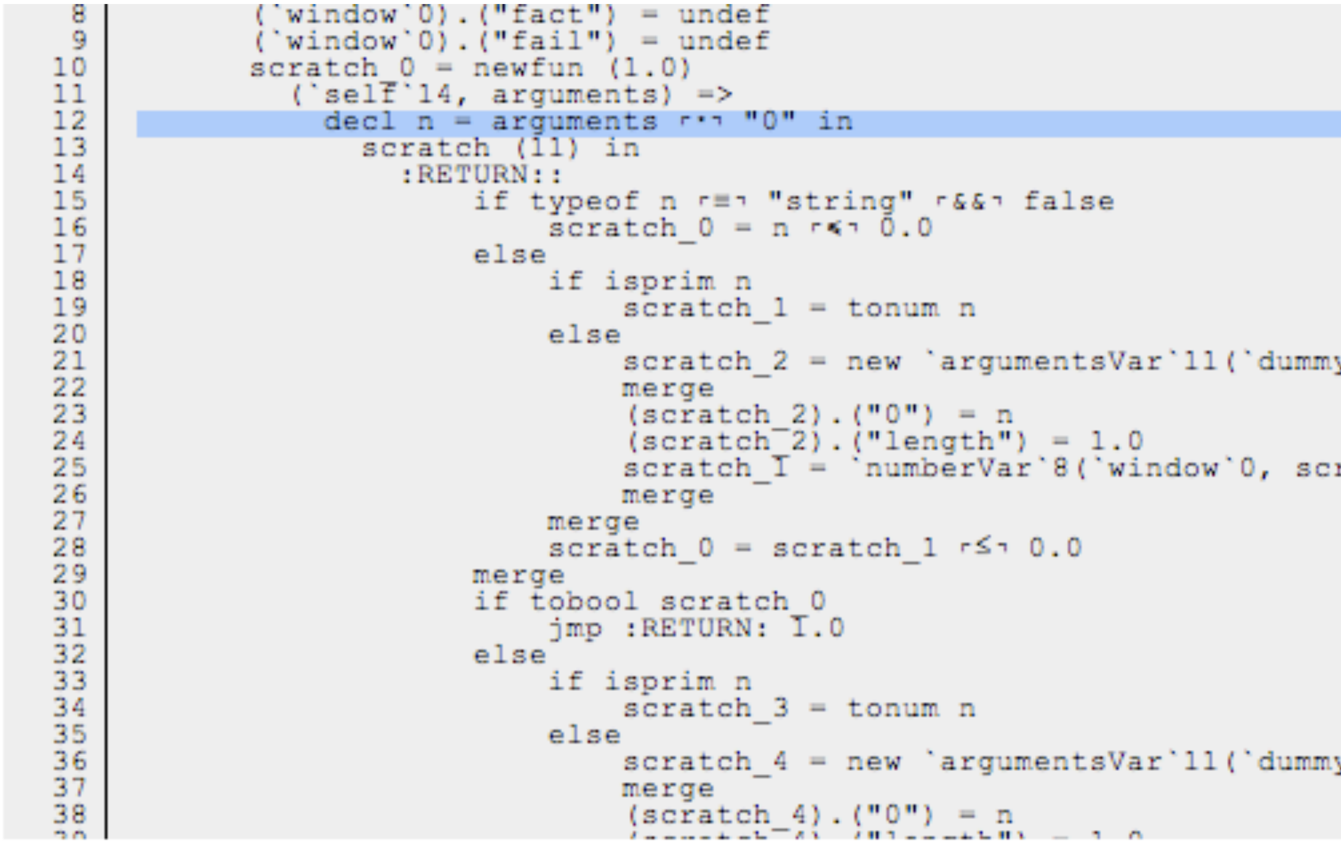
Consider the precise graph…





```
    (`window`0).("fact") = undef
    (`window`0).("fail") = undef
    scratch_0 = newfun (1.0)
      (`self`14, arguments) =>
        decl n = arguments ⌜·⌝ "0" in
          scratch (11) in
            :RETURN::
                if typeof n ⌜=⌝ "string" ⌜&&⌝ false
                    scratch_0 = n ⌜<⌝ 0.0
                else
                    if isprim n
                        scratch_1 = tonum n
                    else
                        scratch_2 = new `argumentsVar`11(`dummy
                        merge
                        (scratch_2).("0") = n
                        (scratch_2).("length") = 1.0
                        scratch_1 = `numberVar`8(`window`0, scr
                        merge
                    merge
                    scratch_0 = scratch_1 ⌜≤⌝ 0.0
                merge
                if tobool scratch_0
                    jmp :RETURN: 1.0
                else
                    if isprim n
                        scratch_3 = tonum n
                    else
                        scratch_4 = new `argumentsVar`11(`dummy
                        merge
                        (scratch_4).("0") = n
                        (scratch_4).("length") = 1.0
```





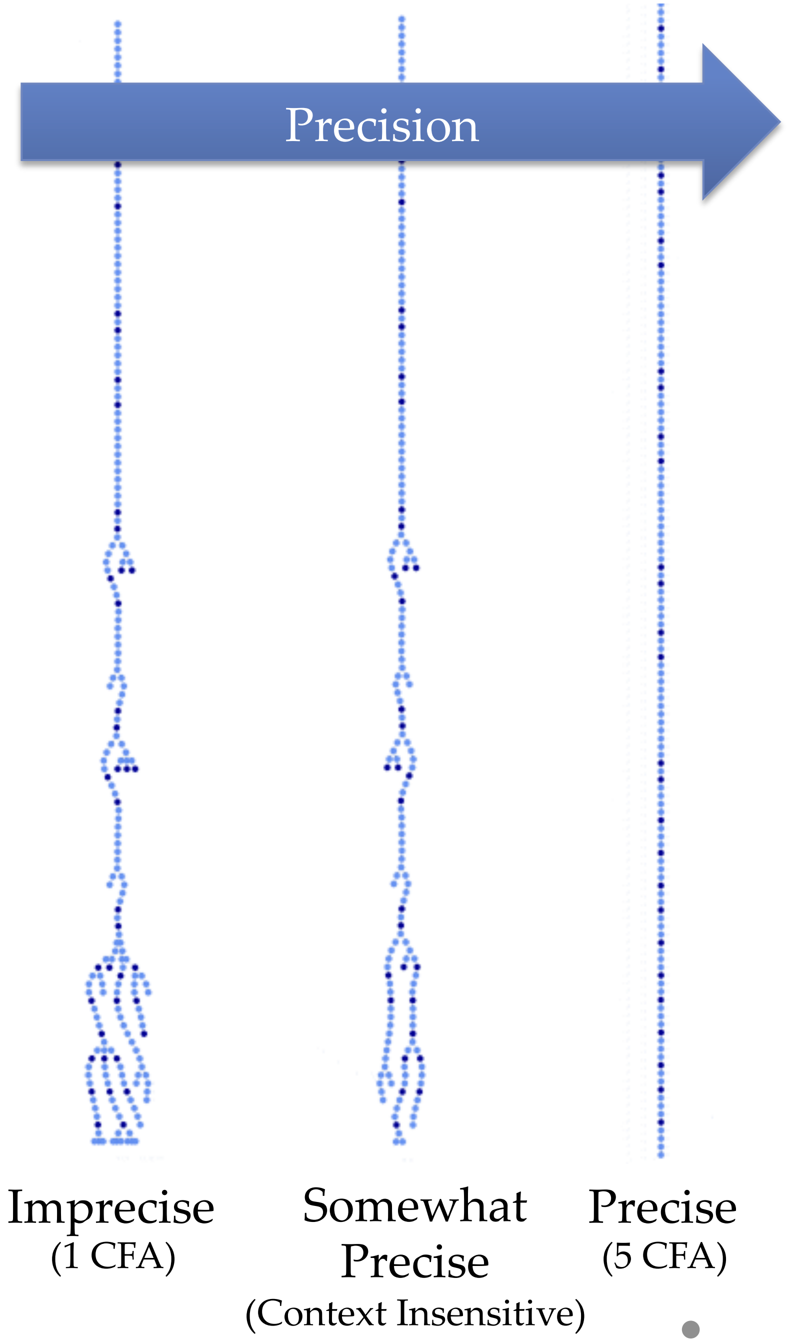
```
(`window`0).("fact") = undef
(`window`0).("fail") = undef
scratch_0 = newfun (1.0)
  (`self`14, arguments) =>
    decl n = arguments ⌜·⌝ "0" in
      scratch (11) in
        :RETURN::
            if typeof n ⌜≡⌝ "string" ⌜&&⌝ false
                scratch_0 = n ⌜≮⌝ 0.0
            else
                if isprim n
                    scratch_1 = tonum n
                else
                    scratch_2 = new `argumentsVar`11(`dummy
                    merge
                    (scratch_2).("0") = n
                    (scratch_2).("length") = 1.0
                    scratch_1 = `numberVar`8(`window`0, scr
                    merge
                merge
                scratch_0 = scratch_1 ⌜≤⌝ 0.0
            merge
            if tobool scratch_0
                jmp :RETURN: 1.0
            else
                if isprim n
                    scratch_3 = tonum n
                else
                    scratch_4 = new `argumentsVar`11(`dummy
                    merge
                    (scratch_4).("0") = n
```

How can we better understand programs?

Which analyses are the most precise?

```
…
=> {FSCI(58250)}
=> {FSCI(58170),FSCI(58240)}
=> {FSCI(58230)}
=> {FSCI(58260)}
=> {FSCI(58160)}
=> {}
=> {FSCI(56360)}
=> {FSCI(56400)}
=> {FSCI(58890)}
=> {FSCI(58930)}
=> {FSCI(58930)}
=> {FSCI(58940)}
60180: DNum:NTop|
DAddr:Set(Address(-69), Address(-30),
Address(-61), Address(-70), Address(-57),
Address(-65), Address(-45), Address(-53),
Address(-59), Address(-46), Address(-58),
Address(-62), Address(-66), Address(-47),
Address(-14), Address(-49), Address(-48),
Address(-50), Address(-72))|DUndef
[success] Total time: 2 s, completed Jul 4,
2013 8:07:56 PM
```

```
...
=> {FSCI(58250)}
=> {FSCI(58170),FSCI(58240)}
=> {FSCI(58230)}
=> {FSCI(58260)}
=> {FSCI(58160)}
=> {}
=> {FSCI(56360)}
=> {FSCI(56400)}
=> {FSCI(58890)}
=> {FSCI(58930)}
=> {FSCI(58930)}
=> {FSCI(58940)}
60180: DNum:NTop|
DAddr:Set(Address(-69), Address(-30),
Address(-61), Address(-70), Address(-57),
Address(-65), Address(-45), Address(-53),
Address(-59), Address(-46), Address(-58),
Address(-62), Address(-66), Address(-47),
Address(-14), Address(-49), Address(-48),
Address(-50), Address(-72))|DUndef
[success] Total time: 2 s, completed Jul 4,
2013 8:07:56 PM
```



tinyurl.com/JSAIVisualizer